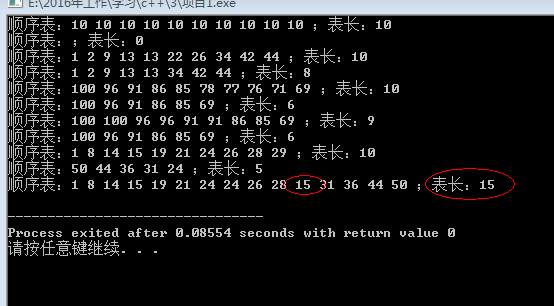
C语言练习代码（3）--有序顺序表

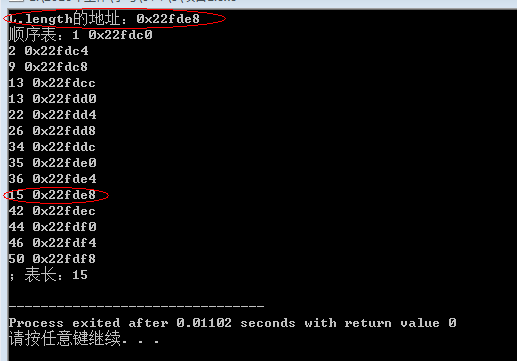
另外说一下,下面的代码目的是熟悉顺序表,并且练习算法,所以不一定是针对问题最合适的算法.

注意:由于一开始MaxSize设置的是10,在将两个有序顺序表合并为一个新的有序表时,出现了错误:第11个位置是最终L.length的值.



因为对应L这个顺序表,其中数组长度为10,第11个位置存储的是L.length的值,所以合并完成后,L.length重新定义的时候,会把第11个位置值覆盖.

请参照下图:很显然L.length的最终长度和第11个位置的值相同,即把之前的值覆盖了.



下面代码我将MaxSize设为了20

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

using namespace std;

#define MaxSize 20

typedef struct{

int data[MaxSize];

int length;

}SqList;

//初始化为一个长度为n的升序顺序表，便于测试

void InitListUp(SqList &L, int n) {

int i;

L.data[0] = 1;

for(i = 1; i < n ; i++) {

int a = rand()%10;

L.data[i] = L.data[i-1] + a;

}

L.length = n;

}

//初始化为一个长度为n的降序顺序表，便于测试

void InitListDown(SqList &L, int n) {

int i;

L.data[0] = 10\*n;

for(i = 1; i < n ; i++) {

int a = rand()%10;

L.data[i] = L.data[i-1] - a;

}

L.length = n;

}

//初始化为一个长度为n的常数序列

void InitListNormal(SqList &L, int n) {

int i;

int a = 10;

for(i = 0; i < n ; i++) {

L.data[i] = a;

}

L.length = n;

}

//将顺序表打印出来

void printList(SqList &L) {

int\* list = L.data;

cout<<"顺序表：";

for(int i = 0; i <L.length; i++) {

cout<<\*(list++)<<" ";

}

cout<<"；表长："<<L.length<<endl;

}

//将顺序表逆置

void Inverse(SqList &L) {

int temp, i;

for(i = 0; i<L.length/2; i++) {

temp = L.data[i];

L.data[i] = L.data[L.length-1-i];

L.data[L.length-1-i] =temp;

}

}

//向顺序表第p个位置中中插入元素e

bool Insert(SqList &L, int p,int e) {

if(p<1||p>L.length+1)

return false;

for(int i = L.length; i >= p; i--) {

L.data[i] = L.data[i-1];

}

L.data[p-1] = e;

L.length++;

return true;

}

//删除有序顺序表中满足s<=x<=t的值

bool DeleBetwST(SqList &L, int s, int t) {

if(L.length == 0)

return false;

if(s>t)

return false;

int p1,p2; //删除位置p1~p2的元素（有序顺序表，删除的元素位置是连续的）

if(L.data[0] > L.data[L.length-1]) { //顺序表是降序的

for(p1 = 0; (p1 < L.length)&&(L.data[p1] > t); p1++);

for(p2 = L.length-1; (p2 > -1)&&(L.data[p2] < s); p2--);

for(; p1 <(L.length-(p2-p1+1)); p1++, p2++) {

L.data[p1] = L.data[p2+1] ;

}

L.length = L.length - (p2-p1+1);

} else if(L.data[0] < L.data[L.length-1]) { //顺序表是升序的

for(p1 = 0; (p1 < L.length)&&(L.data[p1] < s); p1++);

for(p2 = L.length-1; (p2 > -1)&&(L.data[p2] > t ); p2--);

for(; p1 <(L.length-(p2-p1+1)); p1++, p2++) {

L.data[p1] = L.data[p2+1] ;

}

L.length = L.length - (p2-p1+1);

} else { //常数序列

if((L.data[0] >= s)&&(L.data[0] <=t)) { //常数值介于s和t之间 ，全部被删除

L.length = 0;

return true;

} else //常数不在s和t之间，不做任何处理，直接return

return true;

}

}

//删除有序顺序表中重复的元素，使每个元素只出现一次

void DeleRepeat(SqList &L) {

int i = 1, j;

for(j = 1; j < L.length; i++, j++) {

for(;L.data[j] == L.data[j-1];j++);

L.data[i] = L.data[j];

}

L.length = i;

}

//将两个顺序表合成一个新的，并用L返回 ，返回为升序（如果想返回为降序，可以将返回的升序逆置或者稍微修改下逻辑）

void ComBine(SqList &L, SqList &L1, SqList L2) {

if(L1.data[L1.length-1] < L1.data[0]) //如果L不是升序，逆置

Inverse(L1);

if(L2.data[L2.length-1] < L2.data[0]) //如果L1不是升序，逆置

Inverse(L2);

int i, j, k;

for(i = 0,j = 0,k = 0; (i <(L1.length + L2.length-1))&&(j < L1.length)&&(k < L2.length); i++) {

//比较部分，直到某一有序顺序表比较完毕跳出循环。此时有 j = L1.length或者 k = L2.length

if(L1.data[j] <= L2.data[k])

L.data[i] = L1.data[j++];

else

L.data[i] = L2.data[k++];

}

//余下部分处理

if(j == L1.length) {

for(; k < L2.length; k++, i++) {

L.data[i] = L2.data[k];

}

} else if(k == L2.length) {

for(; j < L1.length; j++, i++) {

L.data[i] = L1.data[j];

}

}

L.length = L1.length + L2.length;

}

int main(int argc, char\*\* argv) {

SqList L;

InitListNormal(L,10);

printList(L);

DeleBetwST(L, 5, 15);

printList(L);

InitListUp(L,10);

printList(L);

DeleBetwST(L, 18, 33);

printList(L);

InitListDown(L,10);

printList(L);

DeleBetwST(L, 70, 82);

printList(L); //测 DeleBetwST方法的几种不同情况

Insert(L, 1, L.data[0]);

Insert(L, 3, L.data[2]);

Insert(L, 5, L.data[4]); //使有序顺序表中产生重复的元素，插入过程中要保证有序性

printList(L);

DeleRepeat(L); // 测DeleRepeat

printList(L);

SqList L1,L2;

InitListUp(L1,10);

printList(L1);

InitListDown(L2,5);

printList(L2); //构造两个有序顺序表，一个升序，一个降序

ComBine(L,L1,L2); //测 ComBine

printList(L);

return 0;

}

